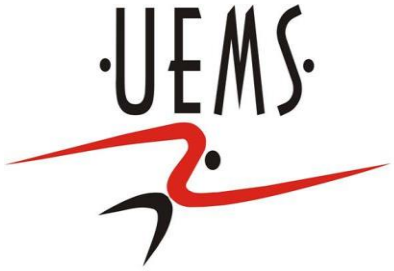


PROLOG

Profa. Mercedes Gonzales Márquez

PROLOG

- O Prolog é uma linguagem de declarativa que usa um fragmento da lógica de 1ª ordem (as Cláusulas de Horn) para representar o conhecimento sobre um dado problema.
- Um programa em Prolog é um “conjunto” de axiomas e de regras de inferência (definindo relações entre objetos) que descrevem um dado problema. A este conjunto chama-se normalmente base de conhecimento.

PROLOG

- A execução de um programa em Prolog consiste na dedução de consequências lógicas da base de conhecimento. O utilizador coloca questões e o “motor de inferência” do Prolog pesquisa a base de conhecimento à procura de axiomas e regras que permitam (por dedução lógica) dar uma resposta.
- O motor de inferência faz a dedução aplicando o algoritmo de resolução de 1ª ordem.

Fatos, Regras e Consultas

- **Programar em Prolog envolve:**

–Declarar alguns **fatos** a respeito de objetos e seus relacionamentos.

–Definir algumas **regras** sobre os objetos e seus relacionamentos.

–Fazer **perguntas (consultas)** sobre os objetos e seus relacionamentos .

Fatos

- **Fatos**: Um fato estabelece um relacionamento incondicional entre objetos de um contexto. Um fato é sempre verdadeiro (verdade incondicional).

-- progenitor(sara,isaque).

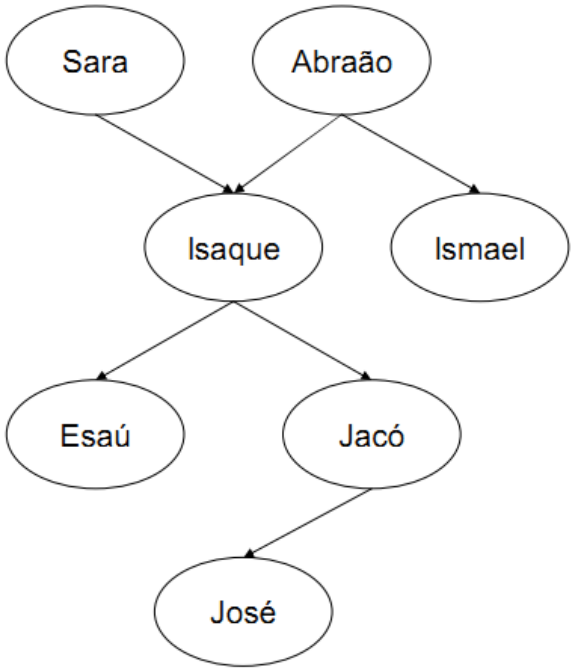
–progenitor(abraão,isaque).

–progenitor(abraão,ismael).

–progenitor(isaque,esaú).

–progenitor(isaque,jacó).

–progenitor(jacó,josé).

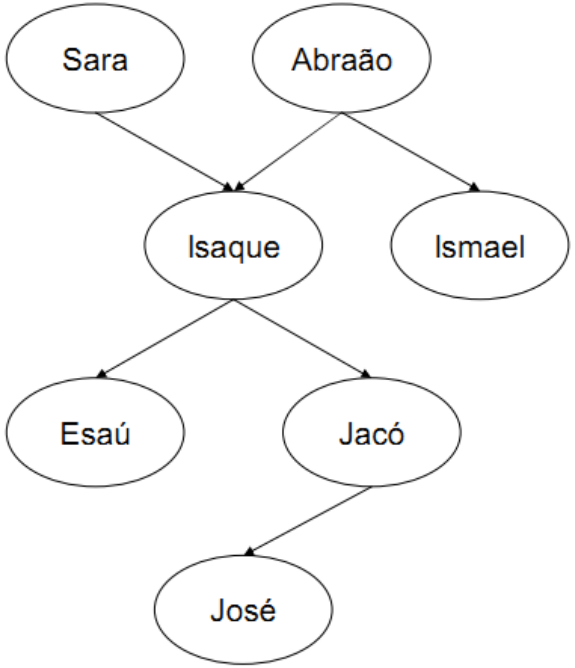


Fatos

- **Fatos**: Um fato estabelece um relacionamento incondicional entre objetos de um contexto. Um fato é sempre verdadeiro (verdade incondicional).

– mulher(sara)

- homem(abraão)
- homem(isaque)
- homem(esaú)
- homem(jacó)



Regras

- **Regras** especificam coisas que são verdadeiras se alguma condição é satisfeita.

- mãe(X,Y) :- mulher(X), progenitor(X,Y).

$\forall X\ \forall Y\ mãe(A,B) \Leftarrow mulher(X) \wedge progenitor(X,Y)$

- pai(X,Y) :- homem(X), progenitor(X,Y).

$\forall A\ \forall B\ pai(A,B) \Leftarrow homem(A) \wedge progenitor(A,B)$

- avo(X,Y) :- progenitor(X,Z), progenitor(Z,Y).

$\forall X\ \forall Y\ avo(X,Y) \Leftarrow \exists Z.\ progenitor(X,Z) \wedge progenitor(Z,Y)$

Cláusulas de Horn

- Cláusulas de Horn são fórmulas da forma

$$p \Leftarrow q1 \wedge q2 \wedge ... \wedge qn$$

representadas em Prolog por

p :- q1, q2, ..., qn

<cabeça de cláusula>:- <corpo de cláusula>

- Os fatos são cláusulas de Horn com o corpo vazio. As variáveis que aparecem nas cláusulas são quantificadas universalmente e o seu âmbito é toda a cláusula.

Cláusulas de Horn

- Mas podemos ver as variáveis que ocorrem apenas no corpo da cláusula (mas não na cabeça), como sendo quantificadas existencialmente dentro do corpo da cláusula.
- As consultas são cláusulas de Horn com cabeça vazia e são um meio de extrair informação de um programa. As variáveis que ocorrem nas consultas são quantificadas existencialmente.

Cláusulas de Horn

- Cláusulas de Horn são fórmulas da forma

$\phi \leftarrow \phi 1, ..., \phi n$, sendo $n>=0$, onde ϕ é uma conclusão e $\phi 1, ..., \phi n$ são premissas (condições).

- Tipos de cláusulas:

Fato.....: $\phi \leftarrow$
Regra.....: $\phi \leftarrow \phi 1, ..., \phi n$
Consulta.....: $\leftarrow \phi 1, ..., \phi n$
Contradição.....: $\leftarrow$ (Cláusula vazia)

Regras e Consultas

- Responder a uma consulta é determinar se a consulta é uma consequência lógica do programa. Responder a uma consulta com variáveis é dar uma instânciação da consulta (representada por uma substituição para as variáveis) que é inferível do programa.

Regras e Consultas

- **Consultas**: Pode-se questionar o Prolog sobre a relação progenitor, por exemplo: **Isaque é o pai de Jacó?**

?- progenitor(abraão,ismael).

Ao que Prolog responderá True

?- progenitor(abraão,moisés).

A resposta será false

Regras e Consultas

- Perguntas mais interessantes também podem ser efetuadas:
- **Quem é o progenitor de Ismael?**
- ?- progenitor(X,josé).

A resposta será

X = jacó

Regras e Consultas

- Ao digitar “;” o motor de inferência faz backtracking. Ou seja, constrói-se uma outra prova (alternativa) para a consulta que é colocada. Essa nova prova pode dar origem a uma outra substituição das variáveis

Regras e consultas

- A pergunta “**Quais os filhos de abraão?**” pode ser escrita como:

?- progenitor(abraão,X).

- Neste caso, há **mais de uma resposta possível**. O Prolog primeiro responde com uma solução:

–X = isaque

- Pode-se requisitar uma **outra solução** (digitando ;) e o Prolog a encontra:

–X = ismael

Regras e Consultas

- Perguntas mais complexas também podem ser efetuadas, tais como: **Quem é o avô de Esaú?**
- Esta pergunta composta pode ser escrita em Prolog como:
- ?- progenitor(Y,esaú), progenitor(X,Y).
- X = abraão
- Y = isaque

Mais regras e consultas

- Outras regras
- filho(Y,X) :- progenitor(X,Y).

Esta cláusula também pode ser lida como: *f*

$\forall X \forall Y \; filho(Y,X) \Leftarrow progenitor(X,Y)$

Para todo X e Y, se X é um progenitor de Y então Y é um filho de X

- irmão(X,Y) :- progenitor(Z,X), progenitor(Z,Y) ?

Mais regras e consultas

- A relação irmão pode ser definida como: ƒ

Para todo X e Y, X é irmão de Y se ambos X e Y têm um progenitor em comum e X é um homem. ‰ Em Prolog: ƒ

irmão(X,Y) :- progenitor(Z,X), progenitor(Z,Y), homem(X). ‰

ƒ?- irmão(X,jacó). ‰ Prolog fornecerá duas respostas ƒ X = esaú ; ƒX = jacó ‰

Assim, Jacó é irmão dele mesmo?

Mais regras e consultas

- De acordo com nossa definição sobre irmãos, a resposta de Prolog é perfeitamente lógica.
- Nossa regra sobre irmãos não menciona que X e Y não devem ser a mesma pessoa se X deve ser irmão de Y.
- Isso nos leva à seguinte regra sobre irmãos: *f* irmão(X,Y) :- progenitor(Z,X), progenitor(Z,Y), homem(X), X\==Y.

Exercícios

- Traduza para Prolog:
- Todo mundo que tem filho é feliz (defina a relação unária feliz) ‰
- Defina a relação irmã.
- Defina a relação neto usando a relação progenitor
- Defina a relação tio(X,Y) em termos das relações progenitor e irmão

Exercícios

- feliz(X) :- progenitor(X,_).
- neto(X,Y) :- progenitor(Y,Z),progenitor(Z,X),homem(X).
- tio(X,Y) :- progenitor(Z,Y),irmão(Z,X),homem(X).

Recursividade

- Regras Recursivas

•Para criar uma relação **ancestral** é necessária a criação de uma regra recursiva:

ancestral(X,Z) :- progenitor(X,Z).

ancestral(X,Z) :- progenitor(X,Y), ancestral(Y,Z).

Recursividade

ancestral(X,Z) :- % caso base

progenitor(X,Z). ‰

ancestral(X,Z) :- % caso recursivo

progenitor(X,Y), ancestral(Y,Z). ‰

Podemos perguntar: quais os descendentes de Sara? *f*

?- ancestral(sara,X). *f*

X = isaque; *f*X = esaú; *f*X = jacó; *f*X = josé

Estruturas

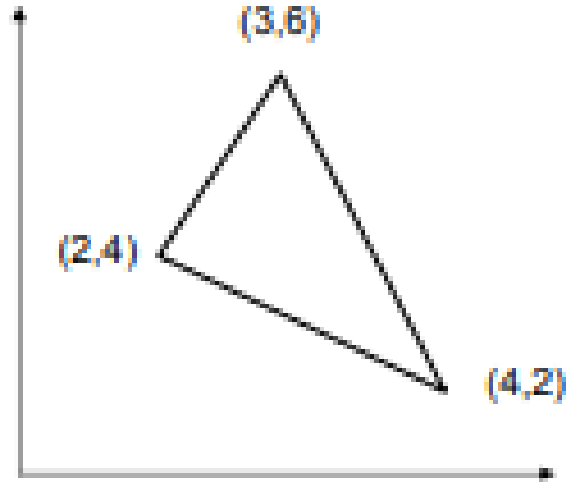
- Objetos estruturados (ou simplesmente estruturas) são objetos de dados que têm vários componentes .
- Cada componente, por sua vez, pode ser uma estrutura. ‰
- Por exemplo, uma data pode ser vista como uma estrutura com três componentes: dia, mês, ano. ‰
- Mesmo possuindo vários componentes, estruturas são tratadas como simples objetos

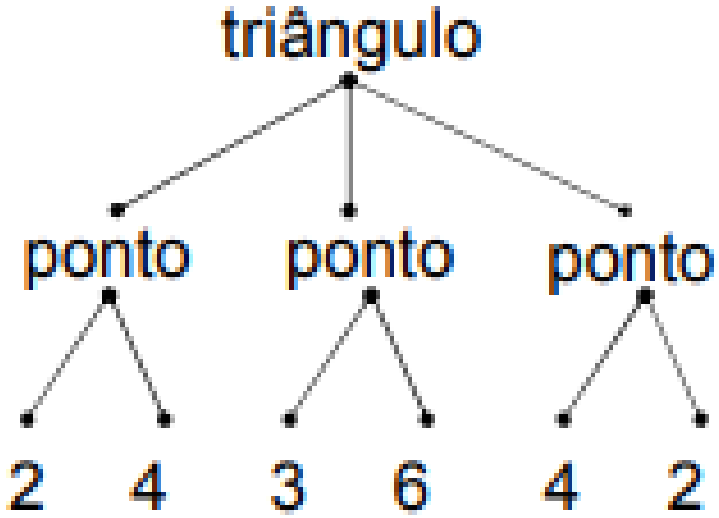
Estruturas

- De forma a combinar componentes em um simples objeto, deve-se escolher um functor (nome de função).
- Um functor para o exemplo da data seria data ‰ Então a data de 3 de novembro de 2025 pode ser escrita como: *f*data(3,novembro,2025)
- Todos os objetos estruturados podem ser representados como árvores ‰ A raiz da árvore é o functor e os filhos da raiz são os componentes .

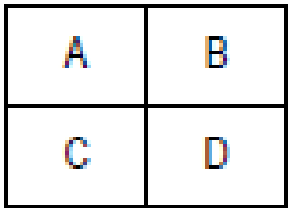
Estruturas

- Por exemplo, o triângulo pode ser representado como triângulo(ponto(2,4),ponto(3,6),ponto(4,2)).





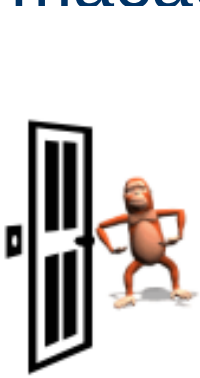
Problema lógico: Colorir um mapa

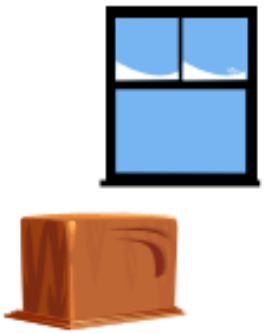


- Para colorir o mapa da Figura com no máximo quatro cores, de modo que regiões adjacentes tenham cores distintas, podemos usar o programa a seguir:
- coloração(A,B,C,D) :-cor(A), cor(B), cor(C), cor(D), A \= B, A \= C, A\=D, B\=C, B \= D, C \= D.

Problema lógico: Macaco & Banana

- Um macaco encontra-se próximo à porta de uma sala. No meio da sala há uma banana pendurada no teto. O macaco tem fome e quer comer a banana mas ela está a uma altura fora de seu alcance. Perto da janela da sala encontra-se uma caixa que o macaco pode utilizar para alcançar a banana.



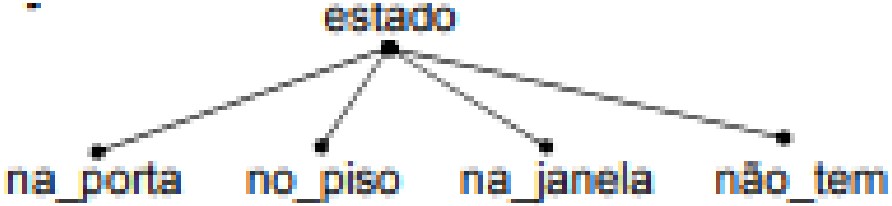


Problema lógico: Macaco & Banana

- O macaco pode realizar as seguintes ações:
  - caminhar no chão da sala; *f*
  - subir na caixa (se estiver ao lado da caixa); *f*
  - empurrar a caixa pelo chão da sala (se estiver ao lado da caixa); *f*
  - pegar a banana (se estiver parado sobre a caixa diretamente embaixo da banana)..

Problema lógico: Macaco & Banana

- É conveniente combinar essas 4 peças de informação em uma estrutura, cujo functor será estado.
- Observe que o estado inicial é determinado pela posição dos objetos:



- O estado final é qualquer estado onde o último componente da estrutura é o átomo tem *f* estado(_,_,_,tem)

Problema lógico: Macaco & Banana

- Possíveis valores para os argumentos da estrutura estado *f*
- 1º argumento (posição horizontal do macaco): na_porta, no_centro, na_janela *f*
- 2º argumento (posição vertical do macaco): no_chão, acima_caixa *f*
- 3º argumento (posição da caixa): na_porta, no_centro, na_janela *f*
- 4º argumento (macaco tem ou não tem banana): tem, não_tem

Problema lógico: Macaco & Banana

- Quais os movimentos permitidos que alteram o mundo de um estado para outro? *ƒ*
  - Pegar a banana *ƒ*
  - Subir na caixa *ƒ*
  - Empurrar a caixa *ƒ*
  - Caminhar no chão da sala ‰
- Nem todos os movimentos são possíveis em cada estado do mundo □ ‘pegar a banana’ somente é possível se o macaco está acima da caixa diretamente abaixo da banana e o macaco ainda não tem a banana ‰

Problema lógico: Macaco & Banana

- Vamos formalizar em Prolog usando a relação move *f*

move(Estado1,Movimento,Estado2) onde Estado1 é o estado antes do movimento, Movimento é o movimento executado e Estado2 é o estado após o movimento.

Problema lógico: Macaco & Banana

- O movimento ‘pegar a banana’ com sua pré-condição no estado antes do movimento pode ser definido por:
move(estado(no_centro,acima_caixa,no_centro,não_tem), pegar_banana, estado(no_centro,acima_caixa,no_centro,tem)). *f*
Este fato diz que após o movimento o macaco tem a banana e ele permanece acima da caixa no meio da sala ‰

Problema lógico: Macaco & Banana

- Vamos expressar o fato que o macaco no chão pode caminhar de qualquer posição horizontal Pos1 para qualquer posição Pos2
  move(estado(Pos1,no_chão,Caixa,Banana),
  caminhar(Pos1,Pos2),
  estado(Pos2,no_chão,Caixa,Banana)). ‰
- De maneira similar, os movimentos ‘empurrar’ e ‘subir’ podem ser especificados

Problema lógico: Macaco & Banana

- A pergunta principal que nosso programa deve responder é: O macaco consegue, a partir de um estado inicial Estado, pegar a banana? ‰

Problema lógico: Macaco & Banana

- Isto pode ser formulado usando o predicado consegue/1 que pode ser formulado baseado em duas observações: ƒPara qualquer estado no qual o macaco já tem a banana, o predicado consegue/1 certamente deve ser verdadeiro; nenhum movimento é necessário ™ consegue(estado(_,_,_,tem)). ƒ

Problema lógico: Macaco & Banana

- Nos demais casos, um ou mais movimentos são necessários; o macaco pode obter a banana em qualquer estado Estado1 se há algum movimento de Estado1 para algum estado Estado2 tal que o macaco consegue pegar a banana no Estado2 (em zero ou mais movimentos) ™
  consegue(Estado1) :-
  move(Estado1,Movimento,Estado2),
  consegue(Estado2).

```
move(estado(no_centro,acima_caixa,no_centro,não_tem), % antes de mover
   pegar_banana,                                      % pega banana
   estado(no_centro,acima_caixa,no_centro,tem) ).     % depois de mover

move(estado(P,no_chão,P,Banana),
    subir,                                            % subir na caixa
    estado(P,acima_caixa,P,Banana) ).
move(estado(P1,no_chão,P1,Banana),
    empurrar(P1,P2),                                  % empurrar caixa de P1 para P2
    estado(P2,no_chão,P2,Banana) ).
move(estado(P1,no_chão,Caixa,Banana),
    caminhar(P1,P2),                                  % caminhar de P1 para P2
    estado(P2,no_chão,Caixa,Banana) ).
consegue(estado(_,_,_,tem)).                          % macaco já tem banana
consegue(Estado1) :-                                  % movimentar e tentar conseguir
move(Estado1,Movimento,Estado2),                      % a banana
consegue(Estado2).
```

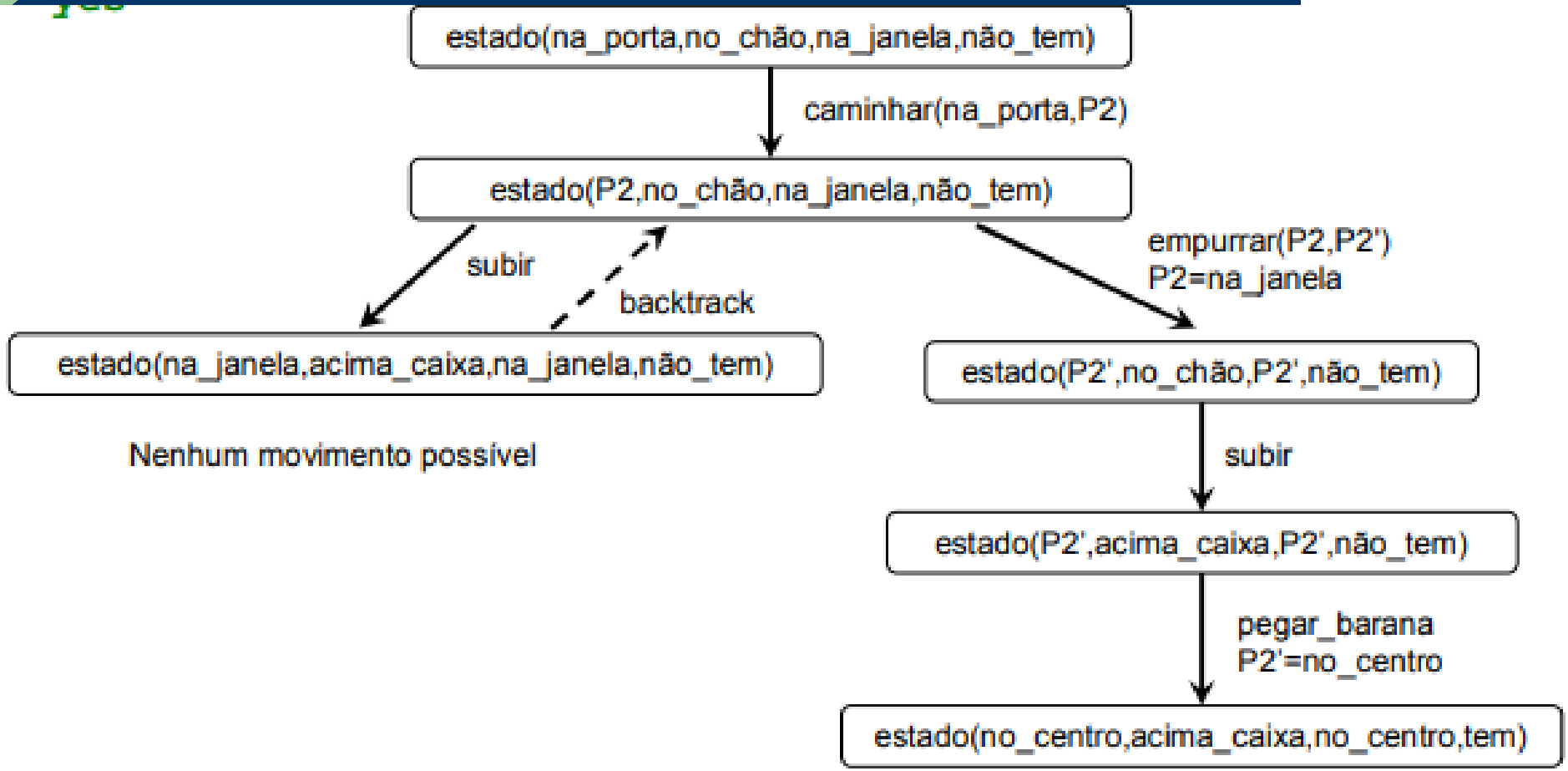
Problema lógico: Macaco & Banana

Consulta

consegue(estado(na_porta,no_chão,na_janela,não_tem)).

yes

Problema lógico: Macaco & Banana



Problema lógico: Macaco & Banana

Exercício:

Faça consultas a partir de outros estados iniciais.

O exemplo Macaco & Banana foi baseado no material do professor José Augusto Baranauskas da USP.